# Non-Clairvoyant Precedence Constrained Scheduling

**Naveen Garg**
Computer Science and Engineering Department, Indian Institute of Technology, Delhi, India
naveen@cse.iitd.ac.in

**Anupam Gupta**
Computer Science Department, Carnegie Mellon University, USA
anupamg@cmu.edu

**Amit Kumar**
Computer Science and Engineering Department, Indian Institute of Technology, Delhi, India
amitk@cse.iitd.ac.in

**Sahil Singla**
Princeton University and Institute for Advanced Study, USA
singla@cs.princeton.edu

**Abstract**

We consider the online problem of scheduling jobs on identical machines, where jobs have precedence constraints. We are interested in the demanding setting where the jobs sizes are not known up-front, but are revealed only upon completion (the *non-clairvoyant* setting). Such precedence-constrained scheduling problems routinely arise in map-reduce and large-scale optimization. For minimizing the total weighted completion time, we give a constant-competitive algorithm. And for total weighted flow-time, we give an $O(1/\varepsilon^2)$-competitive algorithm under $(1+\varepsilon)$-speed augmentation and a natural "no-surprises" assumption on release dates of jobs (which we show is necessary in this context).

Our algorithm proceeds by assigning *virtual rates* to all waiting jobs, including the ones which are dependent on other uncompleted jobs. We then use these virtual rates to decide on the actual rates of minimal jobs (i.e., jobs which do not have dependencies and hence are eligible to run). Interestingly, the virtual rates are obtained by allocating time in a fair manner, using a Eisenberg-Gale-type convex program (which we can solve optimally using a primal-dual scheme). The optimality condition of this convex program allows us to show dual-fitting proofs more easily, without having to guess and hand-craft the duals. This idea of using fair virtual rates may have broader applicability in scheduling problems.

**2012 ACM Subject Classification** Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

**Keywords and phrases** Online algorithms, Scheduling, Primal-Dual analysis, Nash welfare

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2019.63

**Category** Track A: Algorithms, Complexity and Games

**Related Version** `https://arxiv.org/pdf/1905.02133.pdf`

**Funding** This research was supported in part by NSF awards CCF-1536002, CCF-1540541, and CCF-1617790, and the Indo-US Joint Center for Algorithms Under Uncertainty. Sahil Singla was supported in part by the Schmidt Foundation.

## 1 Introduction

We consider the problem of online scheduling of jobs under precedence constraints. We seek to minimize the average weighted flow time of the jobs on multiple parallel machines, in the online *non-clairvoyant* setting. Formally, there are $m$ identical machines, each capable of one unit of processing per unit of time. A set of $[n]$ jobs arrive online. Each job has a processing requirement $p_j$ and a weight $w_j$, and is released at some time $r_j$. If the job finishes at time $C_j$, its flow or response time is defined to be $C_j - r_j$. The goal is to give a preemptive schedule that minimizes the total (or, equivalently, the average) weighted flow-time $\sum_{j\in[n]} w_j \cdot (C_j - r_j)$. The main constraints of our model are the following: (i) the scheduling is done *online*, so the scheduler does not know of the jobs before they are released; (ii) the scheduler is *non-clairvoyant* – when a job arrives, the scheduler knows its weight but *not its processing time* $p_j$. (It is only when the job finishes its processing that the scheduler knows the job is done, and hence knows $p_j$.); And (iii) there are *precedence constraints* between jobs given by a partial order $([n], \prec)$: $j \prec j'$ means job $j'$ cannot be started until $j$ is finished. Naturally, the partial order should respect release dates: if $j \prec j'$ then $r_j \leq r'_j$. (We will require a stronger assumption for some of our results.)

This model for constrained parallelism is a natural one, both in theory and in practice. In theory, this precedence-constrained (and non-clairvoyant!) scheduling model (with other objective functions) goes back to Graham's work on list scheduling [8]. In practice, most languages and libraries produce parallel code that can be modeled using precedence DAGs [20, 1, 9]. Often these jobs (i.e., units of processing) are distributed among some $m$ workstations or servers, either in server farms or on the cloud, i.e., they use identical parallel machines.

## 1.1 Our Results and Techniques

**Weighted Completion Time.** We develop our techniques on the problem of minimizing the *average weighted completion time* $\sum_j w_j C_j$. Our convex-programming approach gives us:

▶ **Theorem 1.1.** *There is a* 10*-competitive deterministic online algorithm for minimizing the average weighted completion time on parallel machines with both release dates and precedences, in the online non-clairvoyant setting.*

For this result, at each time $t$, the algorithm has to know only the partial order restricted to $\{j \in [n] \mid r_j \leq t\}$, i.e., the jobs released by time $t$. The algorithmic idea is simple in hindsight: the algorithm looks at the *minimal* unfinished jobs (i.e., they do not depend on any other unfinished jobs): call them $I_t$. If $J_t$ is the set of (already released and) unfinished jobs at time $t$, then $I_t \subseteq J_t$. To figure out how to divide our processing among the jobs in $I_t$, we write a convex program that fairly divides the time among all jobs in the larger set $J_t$, such that (a) these jobs can "donate" their allocated time to some preceding jobs in $I_t$, and that (b) the jobs in $I_t$ do not get more than 1 unit of processing per time-step.

For this fair allocation, we maximize the (weighted) Nash Welfare $\sum_{j\in J_t} w_j \log R_j$, where $R_j$ is the *virtual rate* of processing given to job $j \in J_t$, regardless of whether it can currently be run (i.e., is in $I_t$). This tries to fairly distribute the virtual rates among the jobs [19], and can be solved using an Eisenberg-Gale-type convex program. (We can solve this convex program in our setting using a simple primal-dual algorithm, see the full version.) The proof of Theorem 1.1 is via writing a linear-programming relaxation for the weighted completion time problem, and fitting a dual to it. Conveniently, the dual variables for the completion time LP naturally fall out of the dual (KKT) multipliers for the convex program!

**Weighted Flow Time.** We then turn to the *weighted flow-time minimization* problem. We first observe that the problem has no competitive algorithm if there are jobs $j$ that depend on jobs released before $r_j$. Indeed, if OPT ever has an empty queue while the algorithm is processing jobs, the adversary could give a stream of tiny new jobs, and we would be sunk. Hence we make an additional *no-surprises* assumption about our instance: when a job $j$ is released, all the jobs having a precedence relationship to $j$ are also released at the same time. In other words, the partial order is a collection of disjoint connected DAGs, where all jobs in each connected component have the same release date. A special case of this model has been studied in [20, 1] where each DAG is viewed as a "hyper-job" and there are no precedence constraints between different hyper-jobs. In this model, we show:

▶ **Theorem 1.2.** *There is an $O(1/\varepsilon^2)$-competitive deterministic non-clairvoyant online algorithm for the problem of minimizing the average weighted flow time on parallel machines with release dates and precedences, under the no-surprises and $(1+\varepsilon)$-speedup assumptions.*

Interestingly, the algorithm for weighted flow-time is almost the same as for weighted completion time. In fact, *exactly* the same algorithm works for both the completion time and flow time cases, if we allow a speedup of $(2+\varepsilon)$ for the latter. To get the $(1+\varepsilon)$-speedup algorithm, we give preference to the recently-arrived jobs, since they have a smaller current time-in-system and each unit of waiting proportionally hurts them more. This is along the lines of strategies like LAPS and WLAPS [7].

## 1.2 The Intuition

Consider the case of unit weight jobs on a single machine. Without precedence constraints, the round-robin algorithm, which runs all jobs at the same rate, is $O(1)$-competitive for the flow-time objective with a 2-speed augmentation. Now consider precedences, and let the partial order be a collection of disjoint chains: only the first remaining job from each chain can be run at each time. We generalize round-robin to this setting by running all minimal jobs simultaneously, but at rates proportional to length of the corresponding chains. We can show this algorithm is also $O(1)$-competitive with a 2-speed augmentation. While this is easy for chains and trees, let us now consider the case when the partial order is the union of general DAGs, where each DAG may have several minimal jobs. Even though the sum of the rates over all the minimal jobs in any particular DAG should be proportional to the number of jobs in this DAG, running all minimal jobs at equal rates does not work. (Indeed, if many jobs depend on one of these minimal jobs, and many fewer depend on the other minimal jobs in this DAG, we want to prioritize the former.)

Instead, we use a convex program to find rates. Our approach assigns a "virtual rate" $R_j$ to each job in the DAG (regardless of whether it is minimal or not). This virtual rate allows us to ensure that even though this job may not run, it can help some minimal jobs to run at higher rates. This is done by an assignment problem where these virtual rates get translated into actual rates for the minimal jobs. The virtual rates are then calculated using Nash fairness, which gives us max-min properties that are crucial for our analysis.

*Analysis Challenges:* In typical applications of the dual-fitting technique, the dual variables for each job encode the *increase in total flow-time* caused by arrival of this job. Using this notion turns out to create problems. Indeed, consider a minimal job of low weight which is running at a high rate (because a large number of jobs depend on it). The increase in overall flow-time because of its arrival is very large. However the dual LP constraints require these dual variables to be bounded by the weights of their jobs, which now becomes difficult to ensure. To avoid this, we define the dual variables directly in terms of the virtual rates of the jobs, given by the convex program.

Having multiple machines instead of a single machine creates new problems. The *actual rates* assigned to any minimal job cannot exceed 1, and hence we have to throttle certain actual rates. Again the versatility of the convex program helps us, since we can add this as a constraint. Arguing about the optimal solution to such a convex program requires dealing with the suitable KKT conditions, from which we can infer many useful properties. We also show in the full version that the optimal solution corresponds to a natural "water-filling" based algorithm.

Finally, we obtain matching results for the case of $(1+\varepsilon)$-speed augmentation. Im et al. [12] gave a general-purpose technique to translate a round-robin based algorithm to a LAPS-like algorithm. In our setting, it turns out that the LAPS-like policy needs to be run on the virtual rates of jobs. Analyzing this algorithm does not follow in a black-box manner (as prescribed by [12]), and we need to adapt our dual-fitting analysis suitably.

## 1.3 Related Work and Organization

**Completion Time.** Minimizing $\sum_j w_j C_j$ on parallel machines with precedence constraints has $O(1)$-approximations in the *offline* setting: Li [16] improves on [11, 18] to give a $3.387+\varepsilon$-approximation. For *related* machines, the precedence constraints make the problem much harder: there is a $O(\log m/\log\log m)$-approximation [16] improving on a prior $O(\log m)$ result [4], and a hardness of $\omega(1)$ under certain complexity assumptions [3]. In the *online* setting, any offline algorithm for (a dual problem to) $\sum_j w_j C_j$ gives an *clairvoyant* online algorithm, losing $O(1)$ factors [11]. Two caveats: it is unclear (a) how to make this algorithm non-clairvoyant, and (b) how to solve the (dual of the) weighted completion time problem with precedences in poly-time.

**Flow Time without Precedence.** To minimize $\sum_j w_j(C_j - r_j)$, strong lower bounds are known for the competitive ratio of any online algorithm even on a single machine [17]. Hence we use speed augmentation [14]. For the general setting of non-clairvoyant weighted flow-time on unrelated machines, Im et al. [13] showed that weighted round-robin with a suitable migration policy yields a $(2+\varepsilon)$-competitive algorithm using $(1+\varepsilon)$-speed augmentation. They gave a general purpose technique, based on the LAPS scheduling policy, to convert any such round-robin based algorithm to a $(1+\varepsilon)$-competitive algorithm while losing an extra $1/\varepsilon$ factor in the competitive ratio. Their analysis also uses a dual-fitting technique [2, 10]. However, they do not consider precedence constraints.

**Flow Time with Precedence.** Much less is known for flow-time problems with precedence constraints. For the offline setting on identical machines, [15] give $O(1)$-approximations with $O(1)$-speedup, even for general delay functions. In the current paper, we achieve a poly$(1/\varepsilon)$-approximation with $(1+\varepsilon)$-speedup for flow-time. Interestingly, [15] show that beating a $n^{1-c}$-approximation for any constant $c \in [0,1)$ requires a speedup of at least the optimal approximation factor of makespan minimization in the same machine environment. However, this lower bound requires different jobs with a precedence relationship to have different release dates, which is something our model disallows. (The full version gives another lower bound showing why we disallow such precedences in the online setting.)

In the *online* setting, [20] introduced the DAG model where each *job* is a directed acyclic graph (of *tasks*) released at some time, and a job/DAG completes when all the tasks in it are finished, and we want to minimize the total *unweighted* flow-time. They gave a $(2+\varepsilon)$-speed $O(\kappa/\varepsilon)$-competitive algorithm, where $\kappa$ is the largest antichain within any job/DAG. [1] show poly$(1/\varepsilon)$-competitiveness with $(1+\varepsilon)$-speedup, again in the non-clairvoyant setting. The case where jobs are entire DAGs, and not individual nodes within DAGs, is captured in our weighted model by putting zero weights for all original jobs, and adding a unit-weight zero-sized job for each DAG which now depends on all jobs in the DAG. Assigning arbitrary weights to individual nodes within DAGs makes our problem quite non-trivial – we need to take into account the structure of the DAG to assign rates to jobs. Another model to capture parallelism and precedences uses *speedup functions* [6, 5, 7]: relating our model to this setting remains an open question.

Our work is closely related to Im et al. [12] who use a Nash fairness approach for completion-time and flow-time problems with multiple resources. While our approaches are similar, to the best of our understanding their approach does not immediately extend to the setting with precedences. Hence we have to introduce new ideas of using virtual rates (and being fair with respect to them), and throttling the induced actual rates at 1. The analyses of [12] and our work are both based on dual-fitting; however, we need some new ideas for the setting with precedences.

**Organization.** The weighted completion time case is solved in §2. A $(2+\varepsilon)$-speedup result for weighted flow-time is in §3. In the full version we improve this to a $(1+\varepsilon)$-speedup. There we also show the need for the "no-surprises" assumption on release dates, how to solve the convex program using a "water-filling" based algorithm, and the missing proofs.

## 2 Minimizing Weighted Completion Time

In this section, we describe and analyze the scheduling algorithm for the problem of minimizing weighted completion time on parallel machines. Recall that the precedence constraints are given by a DAG $G$, and each job $j$ has a release date $r_j$, processing size $p_j$ and weight $w_j$.

### 2.1 The Scheduling Algorithm

We first assume that each of the $m$ machines run at rate 2 (i.e., they can perform 2 units of processing in a unit time). We will show later how to remove this assumption (at a constant loss of competitive ratio). We begin with some notation. We say that a job $j$ is *waiting* at time $t$ (with respect to a schedule) if $r_j \leq t$, but $j$ has not been processed to completion by time $t$. We use $J_t$ to denote the set of waiting jobs at time $t$. Note that at time $t$, the algorithm gets to see the subgraph $G_t$ of $G$ which is induced by the jobs in $J_t$. We say that a job $j$ is *unfinished* at time $t$ if it is either waiting at time $t$, or its release date is at least $t$ (and hence the algorithm does not even know about this job). Let $U_t$ denote the set of unfinished jobs at time $t$. Clearly, $J_t \subseteq U_t$. At time $t$, the algorithm can only process those jobs in $J_t$ which do not have a predecessor in $G_t$ – denote these *minimal* jobs by $I_t$: they are independent of all other current jobs. For every time $t$, the scheduling algorithm needs to assign a rate to each job $j \in I_t$. We now describe how it decides on these rates.

Consider a time $t$. The algorithm considers a bipartite graph $H_t = (I_t, J_t, E_t)$ with vertex set consisting of the minimal jobs $I_t$ on left and the waiting jobs $J_t$ on right. Since $I_t \subseteq J_t$, a job in $I_t$ appears as a vertex on both sides of this bipartite graph. When there is no confusion, we slightly overload terminology by referring to a job as a vertex in $H_t$. The set of edges $E_t$ are as follows: let $j_l \in I_t, j_r \in J_t$ be vertices on the left and the right side respectively. Then $(j_l, j_r)$ is an edge in $E_t$ if and only if there is a directed path from $j_l$ to $j_r$ in the DAG $G_t$.

The following convex program now computes the rate for each vertex in $I_t$. It has variables $z_e^t$ for each edge $e \in E_t$. For each job $j$ on the left side, i.e., for $j \in I_t$, define $L_j^t := \sum_{e \in \partial j} z_e^t$ as the sum of $z_e$ values of edges incident to $j$. Similarly, define $R_j^t := \sum_{e \in \partial j} z_e^t$ for a job $j \in J_t$, i.e., on the right side. The objective function is the Nash bargaining objective function on the $R_j^t$ values, which ensures that each waiting job gets some attention. In the full version we give a combinatorial algorithm to efficiently solve this convex program.

$$\max \sum_{j \in J_t} w_j \ln R_j^t \tag{CP}$$

$$L_j^t = \sum_{j' \in J_t : (j,j') \in E_t} z_{jj'}^t \qquad \forall j \in I_t \tag{1}$$

$$R_j^t = \sum_{j' \in I_t : (j',j) \in E_t} z_{j'j}^t \qquad \forall j \in J_t \tag{2}$$

$$L_j^t \leq 1 \qquad \forall j \in I_t \tag{3}$$

$$\sum_{j \in I_t} L_j^t \leq m \tag{4}$$

$$z_e^t \geq 0 \qquad \forall e \in E_t \tag{5}$$

Let $(\bar{z}^t, \bar{L}^t, \bar{R}^t)$ be an optimal solution to the above convex program. We define the *rate of a job* $j \in I_t$ as being $\bar{L}_j^t$.

Although we have defined this as a continuous time process, it is easy to check that the rates only change if a new job arrives, or if a job completes processing. Also observe that we have effectively combined the $m$ machines into one in this convex program. But assuming that all events happen at integer times, we can translate the rate assignment to an actual schedule as follows. For a time slot $[t, t+1]$, the total rate is at most $m$ (using (4)), so we create $m$ time slots $[t, t+1]_i$, one for each machine $i$, and iteratively assign each job $j$ an interval of length $\bar{L}_j^t$ within these time slots. It is possible that a job may get assigned intervals in two different time slots, but the fact that $\bar{L}_j^t \leq 1$ means it will not be assigned the same time in two different time slots. Further, we will never exceed the slots because of (4). Thus, we can process these jobs in the $m$ time slots on the $m$ parallel machines such that each job $j$ gets processed for $\bar{L}_j^t$ amount of time and no job is processed concurrently on multiple machines. This completes the description of the algorithm; in this, we assume that we run the machines at twice the speed. Call this algorithm $\mathcal{A}$.

The final algorithm $\mathcal{B}$, which is only allowed to run the machines at speed 1, is obtained by running $\mathcal{A}$ in the background, and setting $\mathcal{B}$ to be a slowed-down version of $\mathcal{A}$. Formally, if $\mathcal{A}$ processes a job $j$ on machine $i$ at time $t \in \mathbb{R}_{\geq 0}$, then $\mathcal{B}$ processes this at time $2t$. This completes the description of the algorithm.

## 2.2 A Time-Indexed LP formulation

We use the dual-fitting approach to analyze the above algorithm. We write a time-indexed linear programming relaxation (LP) for the weighted completion time problem, and use the solutions to the convex program (CP) to obtain feasible primal and dual solutions for (LP) which differ by only a constant factor.

We divide time into integral time slots (assuming all quantities are integers). Therefore, the variable $t$ will refer to integer times only. For every job $j$ and time $t$, we have a variable $x_{j,t}$ which denotes the volume of $j$ processed during $[t, t+1]$. Note that this is defined only for $t \geq r_j$. The LP relaxation is as follows:

$$\min \quad \textstyle\sum_{j,t} w_j \cdot \frac{t \cdot x_{j,t}}{p_j} \tag{LP}$$

$$\textstyle\sum_{t \geq r_j} \frac{x_{j,t}}{p_j} \geq 1 \quad \forall j \tag{6}$$

$$\textstyle\sum_j x_{j,t} \leq m \quad \forall t \tag{7}$$

$$\textstyle\sum_{s \leq t} \frac{x_{j,s}}{p_j} \geq \sum_{s \leq t} \frac{x_{j',s}}{p_{j'}} \quad \forall t, j \prec j' \tag{8}$$

The following claim, whose proof is deferred to the full version, shows that it is a valid relaxation.

▷ Claim 2.1. Let *opt* denote the weighted completion time of an optimal off-line policy (which knows the processing time of all the jobs). Then the optimal value of the LP relaxation is at most *opt*.

The (LP) has a large integrality gap. Observe that the LP just imagines the $m$ machines to be a single machine with speed $m$. Therefore, (LP) has a large integrality gap for two reasons: (i) a job $j$ can be processed concurrently on multiple machines, and (ii) suppose we have a long chain of jobs of equal size in the DAG $G$. Then the LP allows us to process all these jobs at the same rate in parallel on multiple machines. We augment the LP lower bound with another quantity and show that the sum of these two lower bounds suffice.

A *chain* $C$ in $G$ is a sequence of jobs $j_1, \ldots, j_k$ such that $j_1 \prec j_2 \prec \ldots \prec j_k$. Define the processing time of $C$, $p(C)$, as the sum of the processing time of jobs in $C$. For a job $j$, define $\mathsf{chain}_j$ as the maximum over all chains $C$ ending in $j$ of $p(C)$. It is easy to see that $\sum_j w_j \cdot (r_j + \mathsf{chain}_j)$ is a lower bound (up to a factor 2) on the objective of an optimal schedule.

We now write down the dual of the LP relaxation above. We have dual variables $\alpha_j$ for every job $j$, and $\beta_t$ for every time $t$, and $\gamma_{s,j\to j'}$

$$\max \quad \sum_j \alpha_j - m \sum_t \beta_t \tag{DLP}$$

$$\alpha_j - w_j \cdot t + \sum_{s \geq t} \Big( \sum_{j \prec j'} \gamma_{s,j\to j'} - \sum_{j' \prec j} \gamma_{s,j'\to j} \Big) \leq p_j \cdot \beta_t \quad \forall j, t \geq r_j \tag{9}$$

$$\alpha_j, \beta_t \geq 0$$

We write the dual constraint (9) in a more readable manner. For a job $j$ and time $s$, let $\gamma^{\text{in}}_{s,j}$ denote $\sum_{j' \prec j} \gamma_{s,j'\to j}$, and define $\gamma^{\text{out}}_{s,j}$ similarly. We now write the dual constraint (9) as

$$\alpha_j - w_j \cdot t + \sum_{s \geq t} \left( \gamma^{\text{out}}_{s,j} - \gamma^{\text{in}}_{s,j} \right) \leq p_j \cdot \beta_t \quad \forall j, t \geq r_j \tag{10}$$

## 2.3 Properties of the Convex Program

We now prove certain properties of an optimal solution $(\bar{z}^t, \bar{L}^t, \bar{R}^t)$ to the convex program (CP). The first property, whose proof is deferred to the full version, is easy to see:

▷ Claim 2.2. If $\sum_{j \in I_t} \bar{L}^t_j < m$, then $\bar{L}^t_j = 1$ for all $j \in I_t$.

We now write down the KKT conditions for the convex program. (In fact, we can use (1) and (2) to replace $\bar{L}^t_j$ and $\bar{R}^t_j$ in the objective and the other constraints.) Then letting $\theta^t_j \geq 0, \eta^t \geq 0, \nu^t_e \geq 0$ be the Lagrange multipliers corresponding to constraints (3), (4) and (5), we get

$$\frac{w_j}{\bar{R}^t_j} = \theta^t_{j'} + \eta^t - \nu^t_e \qquad \forall e = (j', j), j' \in I_t, j \in J_t \tag{11}$$

$$\theta^t_j \, (\bar{L}^t_j - 1) = 0 \qquad \forall j \in I_t \tag{12}$$

$$\eta^t \, (\textstyle\sum_{j \in I_t} \bar{L}^t_j - m) = 0 \tag{13}$$

$$\nu^t_e \cdot \bar{z}^t_e = 0 \qquad \forall e \in E_t \tag{14}$$

We derive a few consequences of these conditions, the proofs are deferred to the full version.

▷ Claim 2.3. Consider a job $j \in J_t$ on the right side of $H_t$. Then $w_j \geq \bar{R}^t_j \cdot \eta^t$.

▷ Claim 2.4. Consider a job $j \in J_t$ on the right side of $H_t$. Suppose $j$ has a neighbor $j' \in I_t$ such that $\bar{L}^t_{j'} < 1$ and $\bar{z}^t_{j'j} > 0$. Then $w_j = \bar{R}^t_j \cdot \eta^t$.

A crucial notion is that of an *active* job:

▶ **Definition 2.5** (Active Jobs). *A job $j \in J_t$ is* active *at time $t$ if it has at least one neighbor in $I_t$ (in the graph $H_t$) running at rate strictly less than 1.*

Let $J^{\text{act}}_t$ denote the set of active jobs at time $t$. We can strengthen the above claim as follows.

▶ **Corollary 2.6.** *Consider an active job $j$ at time $t$. Then $w_j = \bar{R}^t_j \cdot \eta^t$.*

▷ Claim 2.7. $w(J^{\text{act}}_t)/m \leq \eta^t \leq w(J_t)/m$.

## 2.4 Analysis via Dual Fitting

We analyze the algorithm $\mathcal{A}$ first. We define feasible dual variables for (DLP) such that the value of the dual objective function (along with the $\mathsf{chain}_j$ values that capture the maximum processing time over all chains ending in $j$) forms a lower bound on the weighted completion time of our algorithm. Intuitively, $\alpha_j$ would be the weighted completion time of $j$, and $\beta_t$ would be $1/2m$ times the total weight of unfinished jobs at time $t$. Thus, $\sum_j \alpha_j - m \sum_t \beta_t$ would be at 1/2 times the total weighted completion time. This idea works as long as all the machines are busy at any point of time, the reason being that the primal LP essentially views the $m$ machines as a single speed-$m$ machine. Therefore, we can generate enough dual lower bound if the rate of processing in each time slot is $m$. If all machines are not busy, we need to appeal to the lower bound given by the $\mathsf{chain}_j$ values.

We use the notation used in the description of the algorithm. In the graph $H_t$, we had assigned rates $\bar{L}^t_j$ to all the nodes $j$ in $I_t$. Recall that a vertex $j \in J_t$ on the right side of $H_t$ is said to be *active* at time $t$ if it has a neighbor $j' \in I_t$ for which $\bar{L}^t_{j'} < 1$. Otherwise, we say that $j$ is inactive at time $t$. We say that an edge $e = (j_l, j_r) \in E_t$, where $j_l \in I_t, j_r \in J_t$ is *active* at time $t$ if the vertex $j_r$ is active. Let $A_t$ denote the set of active edges in $E_t$. Let $e = (j_l, j_r)$ be an edge in $E_t$. By definition, there is a path from $j_l$ to $j_r$ in $G_t$ – we fix such a path $P_e$. As before, let $C_j$ denote the completion time of job $j$. The dual variables are defined as follows:

- For each job $j$ and time $t$, we define quantities $\alpha_{j,t}$. The dual variable $\alpha_j$ would be equal to $\sum_{t \geq 0} \alpha_{j,t}$. Fix a job $j$. If $t \notin [r_j, C_j]$ we set $\alpha_{j,t}$ to 0. Now, suppose $j \in J_t$. Consider the job $j$ as a vertex in $J_t$ (i.e., right side) in the bipartite graph $H_t$. We set $\alpha_{j,t}$ to $w_j$ if $j$ is active at time $t$, otherwise it is inactive.
- For each time $t$, we set $\beta$ to $1/2m \cdot w(U_t)$ (Recall that $U_t$ is the set of unfinished jobs at time $t$).
- We now need to define $\gamma_{t,j' \to j}$, where $j' \prec j$. If $j$ or $j'$ does not belong to $J_t$, we set this variable to 0. So assume that $j, j' \in J_t$ (and so the edge $(j', j)$ lies in $G_t$). We define

$$\gamma_{t,j' \to j} := \eta^t \cdot \sum_{e: e \in A_t, (j' \to j) \in P_e} \bar{z}^t_e.$$

In other words, we consider all the active edges $e$ in the graph $H_t$ for which the corresponding path $P_e$ contains $(j', j)$. We add up the fractional assignment $\bar{z}^t_e$ for all such edges.

This completes the description of the dual variables.

We first show that the objective function for (DLP) is close to the weighted completion time incurred by the algorithm. The proof is deferred to the full version.

▷ Claim 2.8. The total weighted completion time of the jobs in $\mathcal{A}$ is at most $2(\sum_j \alpha_j - m \cdot \sum_t \beta_t) + \sum_j w_j \cdot (\mathsf{chain}_j + 2r_j)$.

We now argue about feasibility of the dual constraint (9). Consider a job $j$ and time $t \geq r_j$. Since $\alpha_{j,s} \leq w_j$ for all time $s$, $\sum_{s \leq t} \alpha_{j,s} \leq w_j \cdot t$. Therefore, it suffices to show:

$$\sum_{s \geq t} \alpha_{j,s} + \sum_{s \geq t} \left( \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} \right) \leq p_j \cdot \beta_t \tag{15}$$

Let $t_j^\star$ be the first time $t$ when the job $j$ appears in the set $I_t$. This would also be the first time when the algorithm starts processing $j$ because a job that enters $I_t$ does not leave $I_t$ before completion.

▷ Claim 2.9. For any time $s$ lying in the range $[r_j, t_j^\star)$, $\alpha_{j,s} + \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} = 0$.

*Proof.* Fix such a time $s$. Note that $j \notin I_s$. Thus $j$ appears as a vertex on the right side in the bipartite graph $H_s$, but does not appear on the left side. Let $e$ be in active edge in $H_s$ such that the corresponding path $P_e$ contains $j$ as an internal vertex. Then $\bar{z}_e^s$ gets counted in both $\gamma_{s,j}^{\text{out}}$ and $\gamma_{s,j}^{\text{in}}$. There cannot be such a path $P_e$ which starts with $j$, because then $j$ will need to be on the left side of the bipartite graph. There could be paths $P_e$ which end with $j$ – these will correspond to active edges $e$ incident with $j$ in the graph $H_t$ (this happens only if $j$ itself is active). Let $\Gamma(j)$ denote the edges incident with $j$. We have shown that

$$\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} = -\eta^t \cdot \sum_{e \in \Gamma(j) \cap A_s} \bar{z}_e^s. \tag{16}$$

If $j$ is not active, the RHS is 0, and so is $\alpha_{j,s}$. So we are done. Therefore, assume that $j$ is active. Now, $A(s)$ contains all the edges incident with $j$, and so, the RHS is same as $-\eta^t \cdot \bar{R}_j^t$. But then, Corollary 2.6 implies that $-\eta^t \cdot \bar{R}_j^t = -w_j$. Since $\alpha_{j,s} = w_j$, we are done again. ◁

Coming back to inequality (15), we can assume that $t \geq t_j^\star$. To see this, suppose $t < t_j^\star$. Then by Claim 2.9 the LHS of this constraint is same as

$$\sum_{s \geq t_j^\star} \alpha_{j,s} + \sum_{s \geq t_j^\star} \left( \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} \right).$$

Since $\beta_t \geq \beta_{t_j^\star}$ (the set of unfinished jobs can only diminish as time goes on), (15) for time $t$ follows from the corresponding statement for time $t_j^\star$. Therefore, we assume that $t \geq t_j^\star$. We can also assume that $t \leq C_j$, otherwise the LHS of this constraint is 0.

▷ Claim 2.10. Let $s \in [t_j^\star, C_j]$ be such that $j$ is inactive at time $s$. Then $\alpha_{j,s} + \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} \leq \eta^s \cdot \bar{L}_j^s$.

*Proof.* We know that $\alpha_{j,s} = 0$. As in the proof of Claim 2.9, we only need to worry about those active edges $e$ in $H_s$ for which $P_e$ either ends at $j$ or begins with $j$. Since any edge incident with $j$ as a vertex on the right side is inactive, we get (let $\Gamma(j)$ denote the edges incident with $j$, where we consider $j$ on the left side)

$$\alpha_{j,s} + \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} = \eta^s \cdot \sum_{e \in \Gamma(j) \cap A(s)} \bar{z}_e^s \leq \eta^s \cdot \bar{L}_j^s,$$

because $\eta^s \geq 0$ and $\bar{L}_j^s = \sum_{e \in \Gamma(j)} \bar{z}_e^s$. ◁

▷ Claim 2.11. Let $s \in [t_j^\star, C_j]$ be such that $j$ is active at time $s$. Then $\alpha_{j,s} + \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} \le \eta^s \cdot \bar{L}_j^s$.

Proof. The argument is very similar to the one in the previous claim. Since $j$ is active, $\alpha_{j,s} = w_j$. As before we only need to worry about the active edges $e$ for which $P_e$ either ends or begins with $j$. Any edge which is incident with $j$ on the right side (note that there will only one such edge – one the one joining $j$ to its copy on the left side of $H_t$) is active. The following inequality now follows as in the proof of Claim 2.10:

$$\alpha_{j,s} + \gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}} \le w_j + \eta^s \cdot \bar{L}_j^s - \eta^s \cdot \bar{R}_j^s.$$

The result now follows from Corollary 2.6. ◁

We are now ready to show that (15) holds. The above two claims show that the LHS of (15) is at most $\sum_{s=t}^{C_j} \eta^s \cdot \bar{L}_j^s$. Note that for any such time $s$, the rate assigned to $j$ is $\bar{L}_s^j$, and so, we perform $2 \cdot \bar{L}_s^j$ amount of processing on $j$ during this time slot. It follows that $\sum_{s=t}^{C_j} \bar{L}_j^s \le p_j/2$. Now Claim 2.7 shows that $\eta^s \le w(U_s)/m \le w(U_t)/m$, and so we get

$$\sum_{s=t}^{C_j} \eta^s \cdot \bar{L}_j^s \le \frac{p_j \cdot w(U_t)}{2m} = p_j \cdot \beta_t.$$

This shows that (15) is satisfied. We can now prove our algorithm is constant competitive.

▶ **Theorem 2.12.** *The algorithm $\mathcal{B}$ is 10-competitive.*

**Proof.** We first argue about $\mathcal{A}$. We have shown that the dual variables are feasible to (DLP), and so, Claim 2.8 shows that the total completion time of $\mathcal{A}$ is at most $2opt + \sum_j w_j(\mathsf{chain}_j + 2r_j)$, where $opt$ denotes the optimal off-line objective value. Clearly, $opt \ge \sum_j w_j \cdot r_j$ and $opt \ge \sum_j w_j \cdot \mathsf{chain}_j$. This implies that $\mathcal{A}$ is 5-competitive. While going from $\mathcal{A}$ to $\mathcal{B}$ the completion time of each job doubles. ◀

## 3 Minimizing Weighted Flow Time

We now consider the setting of minimizing the total weighted flow time, again in the non-clairvoyant setting. The setting is almost the same as in the completion-time case: the major change is that all jobs which depend on each other (i.e., belong to the same DAG in the "collection of DAGs view" have the same release date). In the full version we show that if related jobs can be released over time then no competitive online algorithms are possible.

As before, let $J_t$ denote the jobs which are *waiting* at time $t$, i.e., which have been released but not yet finished, and let $G_t$ be the union of all the DAGs induced by the jobs in $J_t$. Again, let $I_t$ denote the *minimal* set of jobs in $J_t$, i.e., which do not have a predecessor in $G_t$ and hence can be scheduled.

▶ **Theorem 3.1.** *There exists an $O(1/\varepsilon)$-approximation algorithm for non-clairvoyant DAG scheduling to minimize the weighted flow time on $m$ parallel machines, when there is a speedup of $2 + \varepsilon$.*

The rest of this section gives the proof of Theorem 3.1. The algorithm remains unchanged from §2 (we do not need the algorithm $\mathcal{B}$ now): we write the convex program (CP) as before, which assign rates $\bar{L}_j^t$ to each job $j \in I_t$. The analysis again proceeds by writing a linear programming relaxation, and showing a feasible dual solution. The LP is almost the same as (LP), just the objective is now (with changes in dashed box):

$$\sum_{j,t} w_j \cdot \frac{\boxed{(t-r_j)} \cdot x_{j,t}}{p_j}.$$

Hence, the dual is also almost the same as (DLP): the new dual constraint requires that for every job $j$ and time $t \geq r_j$:

$$\alpha_j + \sum_{s \geq t} \left(\gamma^{\text{out}}_{s,j} - \gamma^{\text{in}}_{s,j}\right) \leq \beta_t \cdot p_j + w_j \boxed{(t-r_j)}. \tag{17}$$

## 3.1 Defining the Dual Variables

In order to set the dual variables, define a total order $\prec$ on the jobs as follows: First arrange the DAGs in order of release dates, breaking ties arbitrarily. Let this order be $D_1, D_2, \ldots, D_\ell$. All jobs in $D_i$ appear before those in $D_{i+1}$ in the order $\prec$. Now for each dag $D_i$, arrange its jobs in the order they complete processing by our algorithm. Note that this order is consistent with the partial order given by the DAG. This also ensures that at any time $t$, the set of waiting jobs in any DAG $D_i$ form a suffix in this total order (restricted to $D_i$).

For a time $t$ and $j \in J_t$, let $\mathbf{I}[j \in J^{\text{act}}_t]$ denote the indicator variable which is 1 exactly if $j$ is active at time $t$. The dual variables are defined as follows:

- For a job $j \in J_t$, we set $\alpha_j := \sum_{t=r_j}^{C_j} \alpha_{j,t}$, where the quantity $\alpha_{j,t}$ as defined as:

$$\alpha_{j,t} := \frac{1}{m}\Big[w_j \cdot \mathbf{I}[j \in J^{\text{act}}_t] \cdot \Big(\sum_{j' \in J_t : j' \preceq j} \bar{R}^t_{j'}\Big) + \bar{R}^t_j \cdot \Big(\sum_{j' \in J^{\text{act}}_t : j' \prec j} w_{j'}\Big)\Big].$$

- The variable $\beta_t := \frac{w(J_t)}{(1+\varepsilon)m}$. Recall that the machines are allowed $2(1+\varepsilon)$-speedup.
- The definition of the $\gamma$ variables changes as follows. Let $(j' \to j)$ be an edge in the DAG $G_t$. Earlier we had considered paths $P_e$ containing $(j' \to j)$ only for the active edges $e$. But now we include *all* edges. Moreover, we replace the multiplier $\eta^t$ by $\eta^t_j$, where $\eta^t_j := \frac{1}{m} \cdot \Big(\sum_{j' \in J_t : j' \preceq j} w_{j'}\Big)$. In other words, we define

$$\gamma_{t, j' \to j} := \eta^t_j \cdot \sum_{e : e \in H_t, (j' \to j) \in P_e} \bar{z}^t_e.$$

In the following sections, we show that these dual settings are enough to "pay for" the flow time of our solution (i.e., have large objective function value), and also give a feasible lower bound (i.e., are feasible for the dual linear program).

## 3.2 The Dual Objective Function

We first show that $\sum_j \alpha_j - m \sum_t \beta_t$ is close to the total weighted flow-time of the jobs. The quantity $\mathsf{chain}_j$ is defined as before. Notice that $\mathsf{chain}_j$ is still a lower bound on the flow-time of job $j$ in the optimal schedule because all jobs of a DAG are simultaneously released. The following claim, whose result is deferred to the full version, shows that the dual objective value is close to the weighted flow time of the algorithm.

▷ Claim 3.2. The total weighted flow-time is at most $\frac{2}{\varepsilon}\Big(\sum_j \alpha_j - m\sum_t \beta_t + \sum_j w_j \cdot \mathsf{chain}_j\Big)$.

## 3.3 Checking Dual Feasibility

Now we need to check the feasibility of the dual constraint (17). In fact, we will show the following weaker version of that constraint:

$$\alpha_j + \boxed{2} \sum_{s \geq t} \left(\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}\right) \leq \beta_t \cdot p_j + \boxed{2} w_j(t - r_j). \tag{18}$$

This suffices to within another factor of 2: indeed, scaling down the $\alpha$ and $\beta$ variables by another factor of 2 then gives dual feasibility, and loses only another factor of 2 in the objective function. We begin by bounding $\alpha_{j,s}$ in two different ways.

▶ **Lemma 3.3.** *For any time $s \geq r_j$, we have $\alpha_{j,s} \leq 2w_j$.*

**Proof.** Consider the second term in the definition of $\alpha_{j,s}$. This term contains $\sum_{j' \in J_s^{\text{act}} : j' \prec j} w_{j'}$. By Corollary 2.6, for any $j' \in J_s^{\text{act}}$ we have $w_{j'} = \bar{R}_{j'}^s \cdot \eta^s$. Therefore,

$$\sum_{j' \in J_s^{\text{act}} : j' \prec j} w_{j'} \quad \leq \quad \eta^s \cdot \sum_{j' \in J_s^{\text{act}} : j' \prec j} \bar{R}_{j'}^s \quad \leq \quad \eta^s \cdot \sum_{j' \in J_s} \bar{R}_{j'}^s.$$

Now we can bound $\alpha_{j,s}$ by dropping the indicator on the first term to get

$$\frac{1}{m} \cdot \left[\left(w_j \cdot \sum_{j' \in J_s : j' \preceq j} \bar{R}_{j'}^s\right) + \bar{R}_j^s \cdot \left(\eta^s \cdot \sum_{j' \in J_s^{\text{act}} : j' \prec j} \bar{R}_{j'}^s\right)\right] \;\leq\; \frac{1}{m} w_j \Big[\sum_{j' \in J_s} \bar{R}_{j'}^s + \sum_{j' \in J_s} \bar{R}_{j'}^s\Big],$$

the last inequality using Claim 2.3. Simplifying, $\alpha_{j,s} \leq \frac{2}{m} \cdot w_j \cdot \sum_{j'' \in I_s} \bar{L}_{j''}^s = 2w_j$. ◀

Here is a slightly different upper bound on $\alpha_{j,s}$.

▶ **Lemma 3.4.** *For any time $s \geq r_j$, we have $\alpha_{j,s} \leq 2\eta_j^s \cdot \bar{R}_j^s$.*

**Proof.** The second term in the definition of $\alpha_{j,s}$ is at most $\eta_j^s \cdot \bar{R}_j^s$, directly using the definition of $\eta_j^s$. For the first term, assume $j$ is active at time $s$, otherwise this term is 0. Now Corollary 2.6 shows that $w_j = \eta^s \cdot \bar{R}_j^s$, so the first term can be bounded as follows:

$$\frac{w_j}{m} \cdot \sum_{j' \in J_s : j' \preceq j} \bar{R}_{j'}^s \quad = \quad \frac{\bar{R}_j^s \cdot \eta^s}{m} \cdot \sum_{j' \in J_s : j' \preceq j} \bar{R}_{j'}^s \quad \overset{\text{(Claim 2.3)}}{\leq} \quad \frac{\bar{R}_j^s}{m} \cdot \sum_{j' \in J_s : j' \preceq j} w_{j'} \quad = \quad \bar{R}_j^s \cdot \eta_j^s,$$

which completes the proof. ◀

To prove (18), we write $\alpha_j = \sum_{s=r_j}^{t-1} \alpha_{j,s} + \sum_{s \geq t} \alpha_{j,s}$, and use Lemma 3.3 to cancel the first summation with the term $2w_j(t - r_j)$. Hence, it remains to prove

$$\sum_{s \geq t} \alpha_{j,s} + 2 \sum_{s \geq t} \left(\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}\right) \leq \beta_t \cdot p_j. \tag{19}$$

Let $t_j^\star$ be the time at which the algorithm starts processing $j$. We first argue why we can ignore times $s < t_j^\star$ on the LHS of (19).

▷ Claim 3.5. Let $s$ be a time satisfying $r_j \leq s < t_j^\star$. Then $\alpha_{j,s} + 2(\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}) \leq 0$.

Proof. While computing $\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}$, we only need to consider paths $P_e$ for edges $e$ in $H_s$ which have $j$ as end-point. Since $j$ does not appear on the left side of $H_s$, this quantity is equal to $-\eta_j^s \cdot \bar{R}_j^s$. The result now follows from Lemma 3.4. ◁

So using Claim 3.5 in (19), it suffices to show

$$\sum_{s \geq \max\{t, t_j^\star\}} \alpha_{j,s} + 2 \sum_{s \geq \max\{t, t_j^\star\}} (\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}) \leq \beta_t \cdot p_j. \tag{20}$$

Note that we still have $\beta_t$ on the right hand side, even though the summation on the left is over times $s \geq \max\{t, t_j^\star\}$. The proof of the following claim is deferred to the full version.

▷ Claim 3.6. Let $s$ be a time satisfying $s \geq \max\{t, t_j^\star\}$. Then $\alpha_{j,s} + 2(\gamma_{s,j}^{\text{out}} - \gamma_{s,j}^{\text{in}}) \leq 2(1+\varepsilon)\beta_t \cdot \bar{L}_j^s$.

Hence, the left-hand side of (20) is at most $2(1+\varepsilon)\beta_t \cdot \sum_{s \geq \max\{t, t_j^\star\}} \bar{L}_j^s$. However, since job $j$ is assigned a rate of $\bar{L}_j^s$ and the machines run at speed $2(1+\varepsilon)$, we get that this expression is at most $p_j \cdot \beta_t$, which is the right-hand side of (20). This proves the feasibility of the dual constraint (18).

**Proof of Theorem 3.1.** In the preceding §3.3 we proved that the variables $\alpha_j/2$, $\beta_t/2$ and $\gamma_{t,j' \to j}$ satisfy the dual constraint for the flow-time relaxation. Since $\sum_j (\alpha_j/2) - m \sum_t (\beta_t/2)$ is a feasible dual, it gives a lower bound on the cost of the optimal solution. Moreover, $\sum_j w_j \cdot \mathsf{chain}_j$ is another lower bound on the cost of the optimal schedule. Now using the bound on the weighted flow-time of our schedule given by Claim 3.2, this shows that we have an $O(1/\varepsilon)$-approximation with $2(1+\varepsilon)$-speedup. ◀

In the full version we show how to use a slightly different scheduling policy that prioritizes the last arriving jobs to reduce the speedup to $(1+\varepsilon)$.

## References

1 Kunal Agrawal, Jing Li, Kefu Lu, and Benjamin Moseley. Scheduling Parallel DAG Jobs Online to Minimize Average Flow Time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 176–189, 2016. doi:10.1137/1.9781611974331.ch14.

2 S. Anand, Naveen Garg, and Amit Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In *SODA'12*, pages 1228–1241. ACM, New York, 2012.

3 Abbas Bazzi and Ashkan Norouzi-Fard. Towards Tight Lower Bounds for Scheduling Problems. In *Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*, pages 118–129, 2015. doi:10.1007/978-3-662-48350-3_11.

4 Fabián A. Chudak and David B. Shmoys. Approximation Algorithms for Precedence-Constrained Scheduling Problems on Parallel Machines that Run at Different Speeds. *J. Algorithms*, 30(2):323–343, 1999. doi:10.1006/jagm.1998.0987.

5 Jeff Edmonds. Scheduling in the Dark. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 179–188, 1999. doi:10.1145/301250.301299.

6 Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. Non-clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics (Extended Abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 120–129, 1997. doi:10.1145/258533.258565.

7 Jeff Edmonds and Kirk Pruhs. Scalably scheduling processes with arbitrary speedup curves. *ACM Trans. Algorithms*, 8(3):Art. 28, 10, 2012. doi:10.1145/2229163.2229172.

8 R. L. Graham. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. doi:10.1002/j.1538-7305.1966.tb01709.x.

9 Robert Grandl, Srikanth Kandula, Sriram Rao, Aditya Akella, and Janardhan Kulkarni. GRAPHENE: packing and dependency-aware scheduling for data-parallel clusters. In *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016.*, pages 81–97, 2016. URL: `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/grandl_graphene`.

10 Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online Primal-Dual for Non-linear Optimization with Applications to Speed Scaling. In *Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*, pages 173–186, 2012. `doi:10.1007/978-3-642-38016-7_15`.

11 Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: off-line and on-line approximation algorithms. *Math. Oper. Res.*, 22(3):513–544, 1997. `doi:10.1287/moor.22.3.513`.

12 Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints. *J. ACM*, 65(1):3:1–3:33, 2018. `doi:10.1145/3136754`.

13 Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. SelfishMigrate: A Scalable Algorithm for Non-clairvoyantly Scheduling Heterogeneous Processors. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 531–540, 2014.

14 Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

15 Janardhan Kulkarni and Shi Li. Flow-time Optimization for Concurrent Open-Shop and Precedence Constrained Scheduling Models. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2018, August 20-22, 2018 - Princeton, NJ, USA*, pages 16:1–16:21, 2018. `doi:10.4230/LIPIcs.APPROX-RANDOM.2018.16`.

16 Shi Li. Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations. In *58th Annual IEEE Symposium on Foundations of Computer Science—FOCS 2017*, pages 283–294. IEEE Computer Soc., Los Alamitos, CA, 2017.

17 Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theorertical Computer Science*, 130(1):17–47, 1994.

18 Alix Munier, Maurice Queyranne, and Andreas S. Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In *Integer programming and combinatorial optimization (Houston, TX, 1998)*, volume 1412 of *Lecture Notes in Comput. Sci.*, pages 367–382. Springer, Berlin, 1998. `doi:10.1007/3-540-69346-7_28`.

19 John F. Nash. The Bargaining Problem. *Econometrica*, 18(2):155–162, 1950. URL: `http://www.jstor.org/stable/1907266`.

20 Julien Robert and Nicolas Schabanel. Non-clairvoyant scheduling with precedence constraints. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 491–500, 2008. URL: `http://dl.acm.org/citation.cfm?id=1347082.1347136`.